Detecting Cryptomining Using Dynamic Analysis

Detecting Cryptomining Using Dynamic Analysis

Domhnall Carlin*,Philip O'Kane†,Sakir Sezer‡ and Jonah Burgess§
Centre for Secure Information Technologies, Queen's University, Belfast, Northern Ireland
Email: *dcarlin05@qub.ac.uk, †p.okane@qub.ac.uk, ‡s.sezer@qub.ac.uk, §jburgess03@qub.ac.uk

***Abstract*—With the rise in worth and popularity of cryptocurrencies, a new opportunity for criminal gain is being exploited and with little currently offered in the way of defence. The cost of mining (i.e. earning cryptocurrency through CPU-intensive calculations that underpin the blockchain technology) can be prohibitively expensive, with hardware costs and electrical overheads previously offering a loss compared to the cryptocurrency gained. Off-loading these costs along a distributed network of machines via malware offers an instantly profitable scenario, though standard Anti-virus (AV) products offer some defences against file-based threats. However, newer fileless malicious attacks, occurring through the browser on seemingly legitimate websites, can easily evade detection and surreptitiously engage the victim machine in computationally-expensive cryptomining (cryptojacking).**

With no current academic literature on the dynamic opcode analysis of cryptomining, to the best of our knowledge, we present the first such experimental study. Indeed, this is the first such work presenting opcode analysis on non-executable files. Our results show that browser-based cryptomining within our dataset can be detected by dynamic opcode analysis, with accuracies of up to 100%. Further to this, our model can distinguish between cryptomining sites, weaponized benign sites, de-weaponized cryptomining sites and real world benign sites. As it is process-based, our technique offers an opportunity to rapidly detect, prevent and mitigate such attacks, a novel contribution which should encourage further future work.

***Index Terms*—Invasive software, Computer Security, Cryptocurrency**

I. Introduction

With iterative developments in the sophistication of malware, traditional AV solutions, based on signature-analysis and vulnerability-monitoring, have been demonstrated to be growingly ineffective [1]. Signature-detection techniques are the most common approach within commercial AV applications [2][3], which rely on analysing a file for pre-learned malware signatures. Any new malicious instances must be captured and analysed for a signature. Considering the rapid evolution of malware, generating and maintaining relevant and up-to-date signatures is an increasingly difficult task. The lag between the malware's appearance 'in the wild' and the counter-measure development, creates an opportunity for the malware to cause serious damage. The effort against malware is constantly behind the curve compared to the onslaught of new malicious code and the resulting damage can be significant [4].

Further to this, there has been a noticeable shift in the actual implementation of malicious software, moving from file-based threats to fileless attacks. As AV technology struggles to keep up, malware creators are becoming increasingly adept at finding new vectors along which to safely usher their code past AV solutions.

978-1-5386-7493-2/18/$31.00 ©2018 IEEE

```
<script src="https://authedmine.com/lib/authedmine.min.js"></script>
<script>
    var miner = new CoinHive.Anonymous
('Ay4wX[redacted]cazpvgxfh[redacted]');
    miner.start(CoinHive.FORCE_EXCLUSIVE_TAB);
</script>
```

Fig. 1. JavaScript implementation of CoinHive (account redacted)

A. Fileless Malware

Fileless malware (also known as non-malware) attacks are defined as malicious attacks on a system, without the use of a file (document, executable, jpg etc) *on disk*. The adversary uses already-existing authorized benign software and processes for malicious purposes, without downloading any files to disk [5]. Such software includes the typical applications an average user would employ daily (e.g. web browser, Office, Flash), and utilities purposely residing in the OS (PowerShell etc). The key point is that the files are not committed to disk, not that files are not downloaded. The attack can simply be triggered by visiting a corrupted website.

B. Cryptomining

Similarly, websites can employ cryptomining software, forcing the user's machine to mine for cryptocurrency (Monero, Bitcoin, Ethereum etc) without their knowledge. This could be viewed as the anti-thesis to ransomware, i.e. criminals are profiting via cryptocurrency by putting others' machines to work, rather than out of action. As no file is copied to disk no signature exists; as the software or processes employed are expected, the attack is practically invisible to endpoint security. The success of these attacks has seen script-based malware increase during the last two years, and as much as 53% of breaches are caused by non-malware attacks [5]. While the detection focus remains on signatures, single time-point or file I/O operations, fileless attacks will continue to grow.

The mining action is typically spawned with trivial JavaScript embedded into a HTML file as shown in Figure 1, though poisoning of third-party plugins can also be used. Easily accessible APIs, such as CoinHive, were made available in late 2017. This can be intentional on the part of the service owner, e.g. as a revenue generating scheme, whether the user is notified or not. Conversely, this code can be inserted maliciously, with neither the user nor owner being aware of the action. Thus the costs are bourne by the service provider and user, and the profits go solely to the miscreant.

The concept of browser-based cryptomining software has been around since 2013, with the proof-of-concept Tidbit created by MIT students as an alternative to in-browser advertising. While the students won a 'most-innovative' hackathon award for the idea, they also received a subpoena from the State of New Jersey in a case which was later dropped subject to conditions. Notably, at the point of settlement, the State conceded that the software did not appear to be intended for malicious purposes [6]. The concept lost favour with the turbulence of cryptocurrencies, but a 1000% rise in Bitcoin value in 2017 was met with a similar surge in interest by those seeking to profit through surreptitious mining. Off-loading the hardware and running expenses of dedicated mining rigs to others, offers overheadless profit potential. This has even fostered competition between cryptomining malware strains, with code found which sought to eject any present miners on the infected host in favour of the newer attacker's [7].

By the beginning of 2018, notable corporate victims of the cryptojacking trend included Starbucks stores in Buenos Aires [8] and YouTube [9]. Almost one billion monthly users of four online video sites were potentially unknowingly used for cryptomining [10], while users of notorious torrenting site The Pirate Bay complained of non-consensual mining [11].

In February 2018, an estimated 4263 websites were found to be inadvertently employing users' CPUs to engage in cryptomining. This occurred due to the breach of a third-party service for website accessibility tools, which was subsequently loaded by the impacted sites, causing a CoinHive miner to be instantiated in each page the plug-in was loaded [12]. As the plug-in was an accessibility aid, many governmental and public agency websites were affected, including uscourts.gov, many council sites in the UK and Ireland, the National Health Service, and, ironically, the UK Information Commissioner's Office. While the attack only lasted an estimated four hours and only generated approximately $24, the potential for escalation of this type of attack across IOT devices is worrying.

C. Mitigations

The UK's National Cyber Security Centre issued advice to developers and administrators following the Feb 2018 breach [13]. The main advice to website administrators centred around the use of Sub-Resource Integrity (SRI) and Content Security Policy (CSP) measures. SRI is a hash-based script checking protocol, allowing a site to validate the integrity of the script being called. CSP is a whitelisting service for third-party script downloads, allowing control over domains from which scripts are permitted. However, SRI is not universally browser-supported, and hashing can be rendered redundant for frequently modified scripts.

A blacklist-based plug-in (NoCoin) was recently added to the Opera and Chrome browser repositories. This allows simple blocking of CoinMining services based on a list of known service providers. However, this only deals with currently known services, and their listed domains.

While closing the browser window may be an intuitive and effective way to stop most browser-based cryptomining, some strains were noted to spawn translucent pop-under windows, which keep the attack alive after the user has seemingly closed the browser down [14]. Similarly, attempting to prevent the JavaScript from running can easily be thwarted by WebASM-enabled threats, featured in newer cryptominer samples [14]. It is apparent that this attack vector is merely at the beginning of its evolution. As witnessed with the explosion in ransomware, the advancement of technologies designed for societal enhancement can be equally and oppositely employed for criminal gain. Yet, to date, the academic literature on the matter is scarce.

D. Contributions

The work presented here offers very clear and notable contributions to the current body of knowledge.

First, to the best of our knowledge, this is the first application of dynamic opcode analysis using a non-executable subject file, traced through a standard benign executable. This opens up an exciting new avenue of investigation, as malware has begun to move away from file-based implementations and the portable executable format.

Second, this is the first such experimental dynamic opcode analysis of cryptomining in the literature, to the best of our knowledge.

Third, our results show that browser-based cryptomining is detectable by dynamic opcode analysis. While other detection techniques are possible, many are easily bypassed. Parsing HTML for the JavaScript necessary for cryptomining execution is easily thwarted with obfuscation, which is readily available. Cycling the mining behaviour on-and-off may avoid host-based detection systems recognising the tell-tale persistent CPU load spike.

Notably, a key feature of malicious cryptomining implementations is the attempt at persisting the behaviour, even when the user has seemingly closed down the browser tab, window or application. As it is process-based, our technique offers an opportunity to detect, prevent and mitigate such attacks, a novel contribution which should encourage further future work.

The remainder of this paper is presented as follows: Section II provides highlights of academic literature in malware analysis, particularly dynamic opcode techniques. Section III describes the methodology employed for the present research. Section IV presents our results, which are discussed in Section V.

II. Related work

Recent research into approaches to the detection of malware, has focused on the runtime behaviour of the code, i.e. what it does at runtime, rather than how it does it. Opcodes (operational codes) are the assembly language instructions directly performed by the CPU. Monitoring and analysing the host operating system's native run-time opcodes provides the opportunity to detect malware operations while bypassing attempts at obfuscation [1].

Santos et al. [15] used a bi-gram representation of opcodes (i.e. consecutively-occurring pairs) in statically-generated datasets. Using ROC-SVM (support vector machine), the authors presented accuracy rates of >85%.

Runwal et al. [16] applied graph techniques to opcodes from Portable Executable (PE) files on the Windows OS, extending [17] in the context of metamorphic malware. Using similarity scores, the model detected families of metamorphic malware from benignware and also other families of metamorphic malware with high accuracy levels.

O'Kane et al. [1] investigated the use of dynamically-yielded opcodes in the detection of obfuscated malware using supervised machine learning. Run-time traces of malicious and benign files were captured using a virtual machine running the file under investigation. The researchers found that 99.5% of variance in the data could be attributed to 8 opcodes, thus reducing the data from the original 150 opcodes.

Carlin et al. [18] used a similar dynamic opcode analysis approach on the classification of around 48,000 malicious executables, while examining the effects of run-length (i.e. number of instructions traced) and n-grams (i.e. combinations of observed opcodes sequences). The authors found that 32k-long opcode run-traces using n=1 offered the peak accuracy (99.05%) with a Random Forest classifier and 10- fold cross-validation. Feature reduction slightly lowered the accuracy to 99.01%, using only 50 opcodes. This demonstates that dynamic opcode analysis, applied to a wide range of malware, can detect the malicious behaviours of unseen malware both speedily and accurately.

Despite the clear advantages of dynamic analysis, to the best of our knowledge no literature currently exists regarding the dynamic execution of cryptomining code, particularly dynamic opcode analysis.

The motivation of the research presented here is to investigate dynamic opcode analysis as a tool for detecting fileless browser-based cryptomining, a first in the literature. Specifically, we sought to address the following research questions:

1) Can dynamic analysis of a legitimate executable provide run traces of malicious and benign non-executable code?
2) Can dynamic opcode analysis be employed to detect the presence of cryptomining code within a browser?
3) Can a machine learning model distinguish benign code from malicious in this context?
4) What quality and quantity of data must be captured in order to generate a suitable model?

III. Methodology

Fig.2 depicts the tracing methodology employed to generate the datasets. Dedicated machines were used for the experiments, using a fresh image of Windows 7 64-bit, an Intel Celeron 2.90GHz G3930 CPU and 4GB RAM. The open source OllyDbg v2 debugger was used to trace the dynamic opcodes of each runtime, with StrongOD v0.4.8.892 used to cloak the running debugger, as per [1]. Firefox 54 was used as the browser to execute all HTML files. Bespoke Python scripts were employed to automate the execution process, which was as follows:

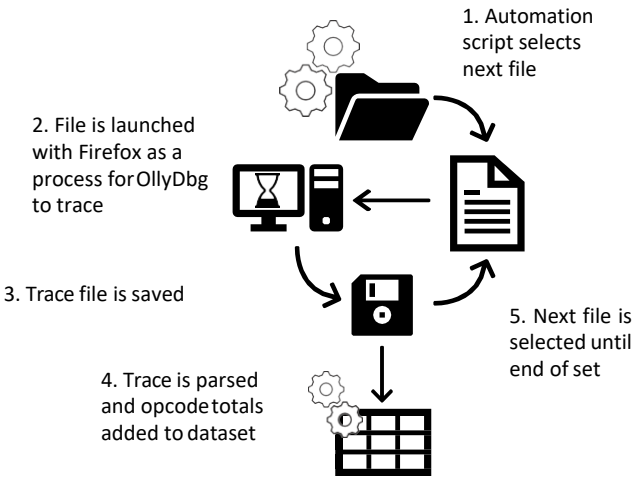

Fig. 2. Tracing process

1) The list of files-to-run was enumerated, and the next HTML file to be launched was passed as an argument to the script.
2) The automation script launched OllyDbg in a pre-configured trace setting. Firefox and the file to be launched were passed to OllyDbg as arguments, enabling OllyDbg to execute Firefox and the file as the debugee, and directly trace the relevant processes.
3) The file was loaded and allowed to continue for 1 minute before tear down. During pilot experiments, this was ample time to allow the file to load correctly and deal with any lag in the debugger.
4) The OllyDbg *trace into* function was used to trace all instructions issued to the CPU and write them to a standard trace file.
5) A bespoke parser was employed to count each of the 610 opcodes in the Intel x86/x64 architecture [19] in each trace, and compile these into a CSV file for use within machine learning frameworks.

This methodology was held constant for creation of all datasets, with the only variable being the input files.

A. Datasets

Four separate datasets were created using the above process, as depicted in Fig. 3.

Cryptomining: 296 samples of CryptoMining HTML files acquired from the VirusShare website [20]. With each having a unique MD5 hash and having been uploaded to the site within the previous two months, all files were checked for the presence of cryptomining code. This formed an observed malicious dataset.

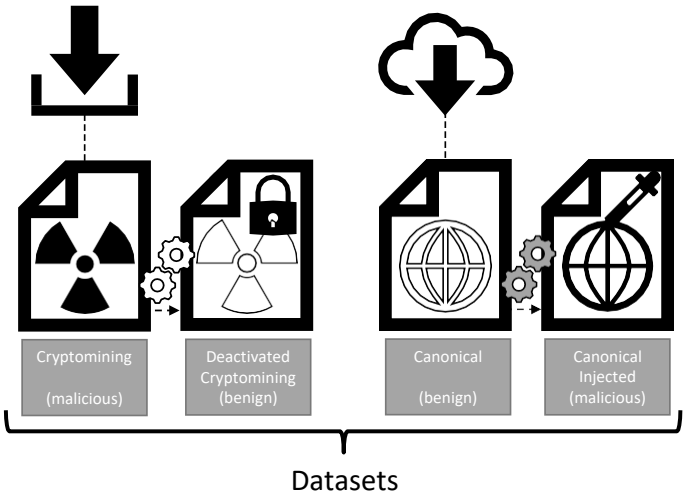

Fig. 3. Datasets derived from source data

Deactivated cryptomining: 194 randomly-chosen cryptomining files were deactivated by removing the relevant `start()` function call, with all other aspects of the file retained. This formed a synthetic benign dataset.

Canonical: A canonical (i.e. legitimately observed) benign dataset was created using the OpenDNS Random Sample List [21], a random sample of 10,000 domain names checked as malware-free. The first 359 sites were chosen.

Canonical injected: 57 weaponized versions of randomly-selected canonical files were created by injecting the malicious cryptomining code, taken from the cryptomining dataset, into the benign files. This created a synthetically malicious dataset, simulating the code which would be experienced by injection attacks on legitimate websites, as mentioned in the introduction to this paper.

B. Machine Learning

The Random Forest (RF) [22] machine learning algorithm was used, as implemented in WEKA 3.9, for all classification tasks. RF is an ensemble learner, combining the decisions of multiple small learners (decision trees). RF varies from traditional tree-based learners, in that each node uses a random number of features to decide the parameter, improving noise immunity and reducing the tendency to overfit. From an implementation point of view, RF functions well with large datasets, i.e. feature vector size and instance quantity. It is capable of parallelization [22], and is also highly recommended for imbalanced data [23].

IV. Results

A. Experiment 1: Distinguishing cryptomining from deactivated cryptomining

A RandomForest model was built and tested on the cryptomining and deactivated cryptomining datasets using 10-fold cross-validation. In total, 490 classifications were attempted, and the model correctly identified 100% of the test instances (194 Benign, 296 Malicious). While this was an impressive result, the websites used to create the benign test set could be viewed as atypical of the average website visited, i.e. only created for the purposes of operating the cryptomining process, thus diminishing the potential application of this model in a real world setting.

TABLE I
Machine learning metrics for the enhanced model, built and tested using 10F CV.

	TP	FP	Prec	Recall	F	MCC	ROC	PRC
Malicious	0.997	0.014	0.987	0.997	0.992	0.983	1	1
Benign	0.986	0.003	0.997	0.986	0.991	0.983	1	1
Avg.	0.992	0.009	0.992	0.992	0.992	0.983	1	1

B. Experiment 2: Model validation against real-world data

Experiment two sought to increase the difficulty of the problem by testing the model against unseen real-world data i.e. the canonical set. A subset of the canonical set was used as an unary-class testing set. The model attempted 259 classifications, and achieved an accuracy of 0%, labelling every instance as malicious. This indicated that the model, as trained on the cryptomining and deactivated cryptomining samples, could not detect the new benign real-world samples.

C. Experiment 3: Model enhancement incorporating real-world data

In order to assess the potential of the technique to be successfully applied outside of the malicious and benign cryptomining data, experiment three exposed the model to the canonical dataset for training. Using the cryptomining, deactivated cryptomining, and canonical datasets, a new model was built and tested using 10-fold cross-validation. Fig. 4 shows the confusion matrix of the results, and Table I depicts the machine learning metrics. Overall accuracy (i.e. correctly classified instances) was 99.15% of 589 attempted, with a true positive (TP) rating of 99.7% and true negative (TN) of 98.6%. The averaged F_1 measure was 0.992, indicating balance between both precision and recall. Matthews' Correlation Coefficient (MCC) was 0.983, suggesting a very strong positive correlation between classifications and observations. The two ratio curve metrics precision-recall curve (PRC) and Receiver Operating Characteristic Curve (ROC) show perfect results.

D. Experiment 4: Enhanced model tested against real-world and simulated data

To examine if this technique could differentiate between benign and cryptomining data, both synthetic and observed, all datasets were combined to provide an overall training and test set. A model was built and evaluated using 10-fold cross validation across all data points (351 malicious and 553 benign). The classifier achieved an accuracy of 99.89%,

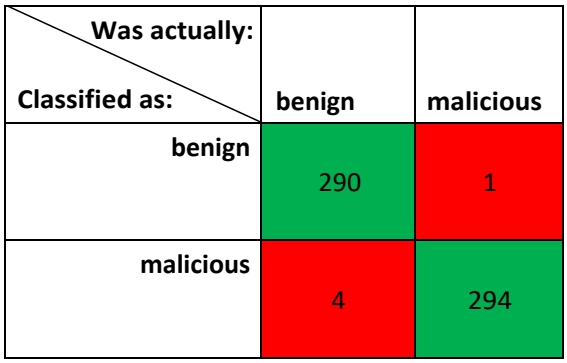

Fig. 4. Confusion matrix for the enhanced model, built and tested using 10f CV.

TABLE II
MACHINE LEARNING METRICS FOR EXPERIMENT 4

Class	TP	FP	Prec	Recall	F	MCC	ROC	PRC
Malicious	0.997	0	1	0.997	0.999	0.998	0.998	0.998
Benign	1	0.003	0.998	1	0.999	0.998	0.998	0.998
Avg.	0.999	0.002	0.999	0.999	0.999	0.998	0.998	0.998

with 903 of the 904 classification attempts being successful. One malicious file was misclassified as benign. On further inspection, the single instance that was misclassified was actually a null trace i.e. all opcodes were recorded as zero. All metrics, as depicted in Table II, showed excellent classification performance, even with the single FN rating.

E. Experiment 5: Further investigation of the enhanced model as a four-class problem

With the extremely accurate classifier performance in Experiment 4, this experiment sought to challenge the classifier further. The previous experiment resolved all data into either benign or malicious (i.e. a two-class problem). For the present experiment, data were labelled under their original dataset into 4 categories (Cryptomining, Deactivated Cryptomining, Canonical, Canonical Injected), and a 4-class problem was presented to a new model, which was again built on all data under 10-fold cross-validation.

The classifier performed with exactly the same accuracy as in Experiment 4, with a single misclassification out of 904 attempts (99.89% accuracy). This solitary instance was the same as with Experiment 4, however it was specifically misclassified as a canonical (benign) file rather than the overall benign class, as shown in Figure 5. The PRC ratio metric ranged from 99.7% (canonical) to 100%, due to the misclassification, which shows excellent performance. Table III lists the full set of machine learning metrics for this experiment.

V. CONCLUSIONS

The results presented in this paper demonstrate that dynamic opcode tracing is extremely effective at detecting cryptomining behaviours in the sample set within a browser.

TABLE III
MACHINE LEARNING METRICS FOR EXPERIMENT 5

Class	TP	FP	Prec	Recall	F	MCC	ROC	PRC
Crypto-deactivated	1	0	1	1	1	1	1	1
Canonical	1	0.002	0.997	1	0.999	0.998	0.999	0.997
Cryptominer	0.997	0	1	0.997	0.998	0.997	0.998	0.998
Canonical Injected	1	0	1	1	1	1	1	1
Avg.	0.999	0.001	0.999	0.999	0.999	0.998	0.999	0.998

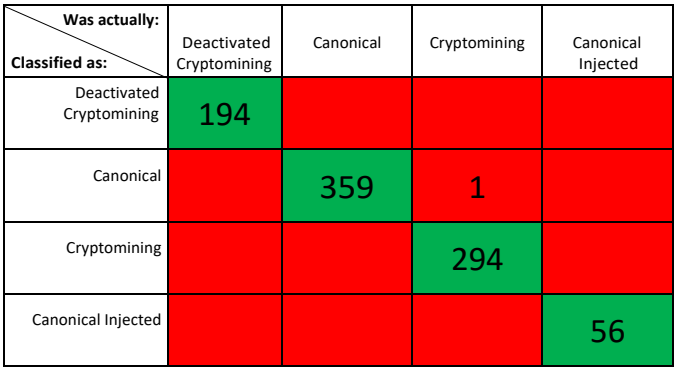

Fig. 5. Confusion matrix for Experiment 5

Our model can distinguish cryptomining-enabled HTML files executed in the browser, from the same files with the cryptomining behaviours deactivated, at high speed and with extremely high levels of accuracy. However, it is clear from Experiments 2 and 3 that training solely on deactivated and cryptomining files does not adequately build a model which is extrapolatable against real-world data. This has obvious implications for any implementation of the techniques described here. Incorporating canonical benign data into the training set fortifies the model, enabling it to distinguish between malicious and benign traces with >99% accuracy, including real-world canonical data.

The key achievement of the results presented in Experiments 4 and 5 is the ability to classify benign from malicious when trained and tested on all four categories, with a solitary misclassification out of greater than 900 attempts. When investigated further, the model is able to detect each of the four categories of data with 99.9% accuracy. Indeed, the only misclassified instance was one in which the trace did not adequately run. As no data cleansing or pre-processing was employed, this anomalous instance was maintained within the dataset, as a representation of what challenges an actual implementation may encounter. The fact that a misclassified instance was the result of a non-trace (i.e. the file detonation failed), serves to reinforce the findings that dynamic opcode tracing offers a very powerful ability to detect cryptomining behaviour within a browser.

While the datasets used for this initial set of experiments were limited in size and scope, our future work will involve a large scale analysis of a greater volume of web pages. This dynamic-based research fits in tandem with our static approaches, which are in the experimental phase. Lastly, we

will further investigate this work by implementing the opcode analysis within the browser, offering a direct approach to detecting malicious activity.

REFERENCES

[1] P. OKane, S. Sezer, K. McLaughlin, and E. G. Im, "Svm training phase reduction using dataset feature filtering for malware detection," *IEEE transactions on information forensics and security*, vol. 8, no. 3-4, pp. 500–509, 2013.

[2] I. Santos, Y. K. Penya, J. Devesa, and P. G. Bringas, "N-grams-based file signatures for malware detection." *ICEIS (2)*, vol. 9, pp. 317–320, 2009.

[3] S. Vemparala, F. D. Troia, V. A. Corrado, T. H. Austin, and M. Stamp, "Malware detection using dynamic birthmarks," in *IWSPA 2016 - Proceedings of the 2016 ACM International Workshop on Security and Privacy Analytics, co-located with CODASPY 2016*, 2016, pp. 41–46.

[4] P. O'Kane, S. Sezer, and K. McLaughlin, "Obfuscation: The hidden malware," *Security Privacy, IEEE*, vol. 9, no. 5, pp. 41–47, 2011.

[5] M. Viscuso, "What is a non-malware (or fileless) attack?" [Online]. Available: https://www.carbonblack.com/2017/02/10/non-malware-fileless-attack/

[6] "Rubin v. new jersey (tidbit)," Electronic Frontier Foundation, Tech. Rep. [Online]. Available: https://www.eff.org/cases/rubin-v-new-jersey-tidbit

[7] X. Mertens, "The crypto miners fight for cpu cycles," Tech. Rep. [Online]. Available: https://isc.sans.edu/forums/diary/The+Crypto+Miners+Fight+For+CPU+Cycles/23407/

[8] L. Kelion, "Starbucks cafe's wi-fi made computers mine crypto-currency," Tech. Rep. [Online]. Available: http://www.bbc.co.uk/news/technology-42338754

[9] "Malvertising campaign abuses google's doubleclick to deliver cryptocurrency miners," Trend Micro, Tech. Rep. [Online]. Available: https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick-to-deliver-cryptocurrency-miners/

[10] "Crypto-streaming strikes back," AdGuard Research, Tech. Rep. [Online]. Available: https://blog.adguard.com/en/crypto-streaming-strikes-back/

[11] H. Lau, "Browser-based cryptocurrency mining makes unexpected return from the dead," Tech. Rep. [Online]. Available: https://www.symantec.com/blogs/threat-intelligence/browser-mining-cryptocurrency

[12] S. Helme, "Protect your site from cryptojacking with csp + sri." [Online]. Available: https://scotthelme.co.uk/protect-site-from-cryptojacking-csp-sri/

[13] "Ncsc advice: Malicious software used to illegally mine cryptocurrency." [Online]. Available: https://www.ncsc.gov.uk/guidance/ncsc-advice-malicious-software-used-illegally-mine-cryptocurrency

[14] J. Segura, "Persistent drive-by cryptomining coming to a browser near you," Tech. Rep. [Online]. Available: https://blog.malwarebytes.com/cybercrime/2017/11/persistent-drive-by-cryptomining-coming-to-a-browser-near-you/

[15] I. Santos, F. Brezo, B. Sanz, C. Laorden, and P. G. Bringas, "Using opcode sequences in single-class learning to detect unknown malware," *IET information security*, vol. 5, no. 4, pp. 220–227, 2011.

[16] N. Runwal, R. M. Low, and M. Stamp, "Opcode graph similarity and metamorphic detection," *Journal in Computer Virology*, vol. 8, no. 1-2, pp. 37–52, 2012.

[17] B. Anderson, D. Quist, J. Neil, C. Storlie, and T. Lane, "Graph-based malware detection using dynamic analysis," *Journal in Computer Virology*, vol. 7, no. 4, pp. 247–258, 2011.

[18] D. Carlin, P. O'Kane, and S. Sezer, *Dynamic Analysis of Malware using Run Time Opcodes*, 1st ed., ser. Data Analytics and Decision Support for Cybersecurity - Trends, Methodologies and Applications. Springer, 2017.

[19] K. Lejska, "X86 opcode and instruction reference." [Online]. Available: http://ref.x86asm.net/

[20] J.-M. Roberts, *VirusShare.com*, 2014.

[21] OpenDNS, "Public domain lists." [Online]. Available: https://github.com/opendns/public-domain-lists

[22] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[23] T. M. Khoshgoftaar, M. Golawala, and J. V. Hulse, "An empirical study of learning from imbalanced data using random forest," in *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, vol. 2. IEEE, 2007, pp. 310–317.